A Decade of Yaws

Steve Vinoski
Architect, Basho Technologies

Erlang Factory London 2011
9 June 2011

@stevevinoski
http://steve.vinoski.net/

What We'll Cover

- Some Yaws history
- Yaws community
- Some Yaws features
- Performance discussion
- Yaws future

What is Yaws?

- "Yet Another Web Server" — an HTTP 1.1 web server
- Brainchild of Claes "Klacke" Wikström, who also created Erlang features such as
  - bit syntax
  - dets
  - Mnesia
  - Distributed Erlang
- Yaws is known for years of reliability and stability
- Current version: 1.90, released 26 May 2011

Why Yaws?

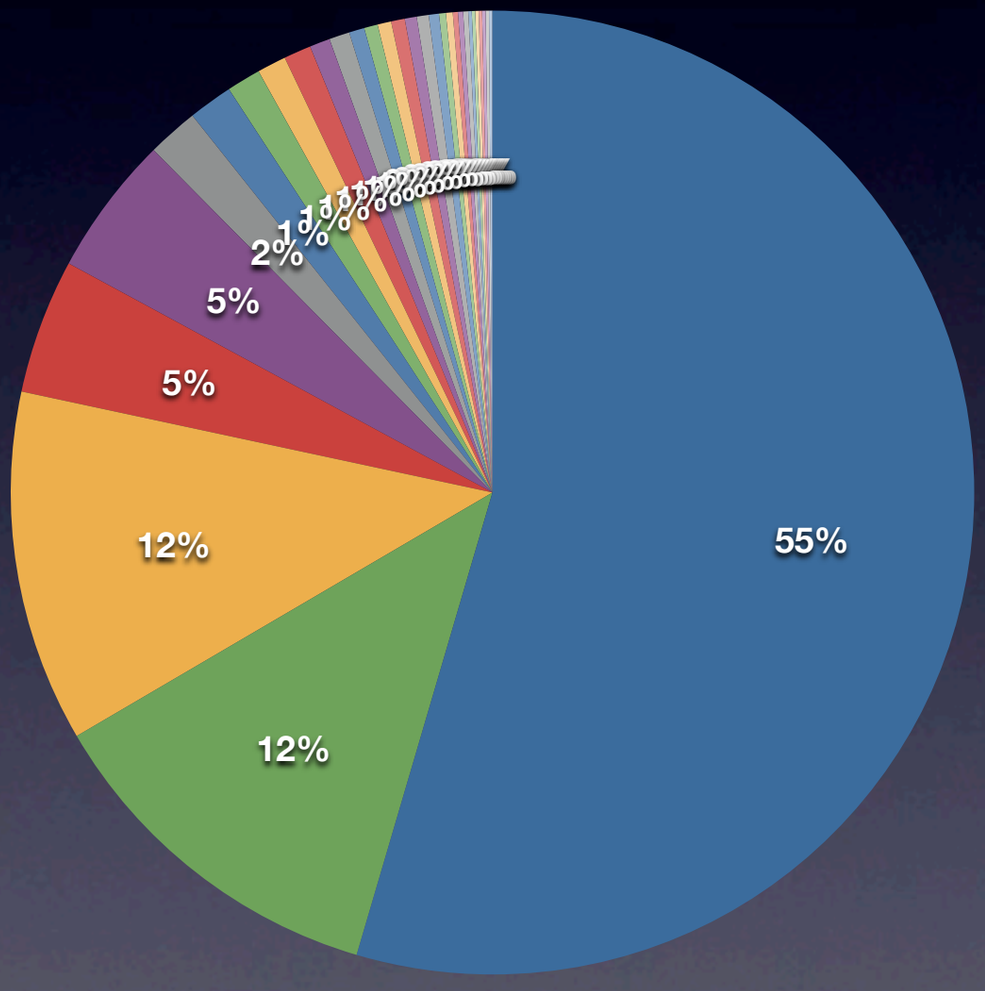
- In 2001 Klacke was in a floorball league and needed a way for players to sign up on the web
- He was horrified by the LAMP stack and PHP
- So he wrote Yaws, but never finished the floorball site :-)

Yaws Community

Website and Email

- Website: http://yaws.hyber.org
- Mailing list: erlyaws-list@lists.sourceforge.net
- List archives: http://sourceforge.net/mailarchive/forum.php?forum_name=erlyaws-list

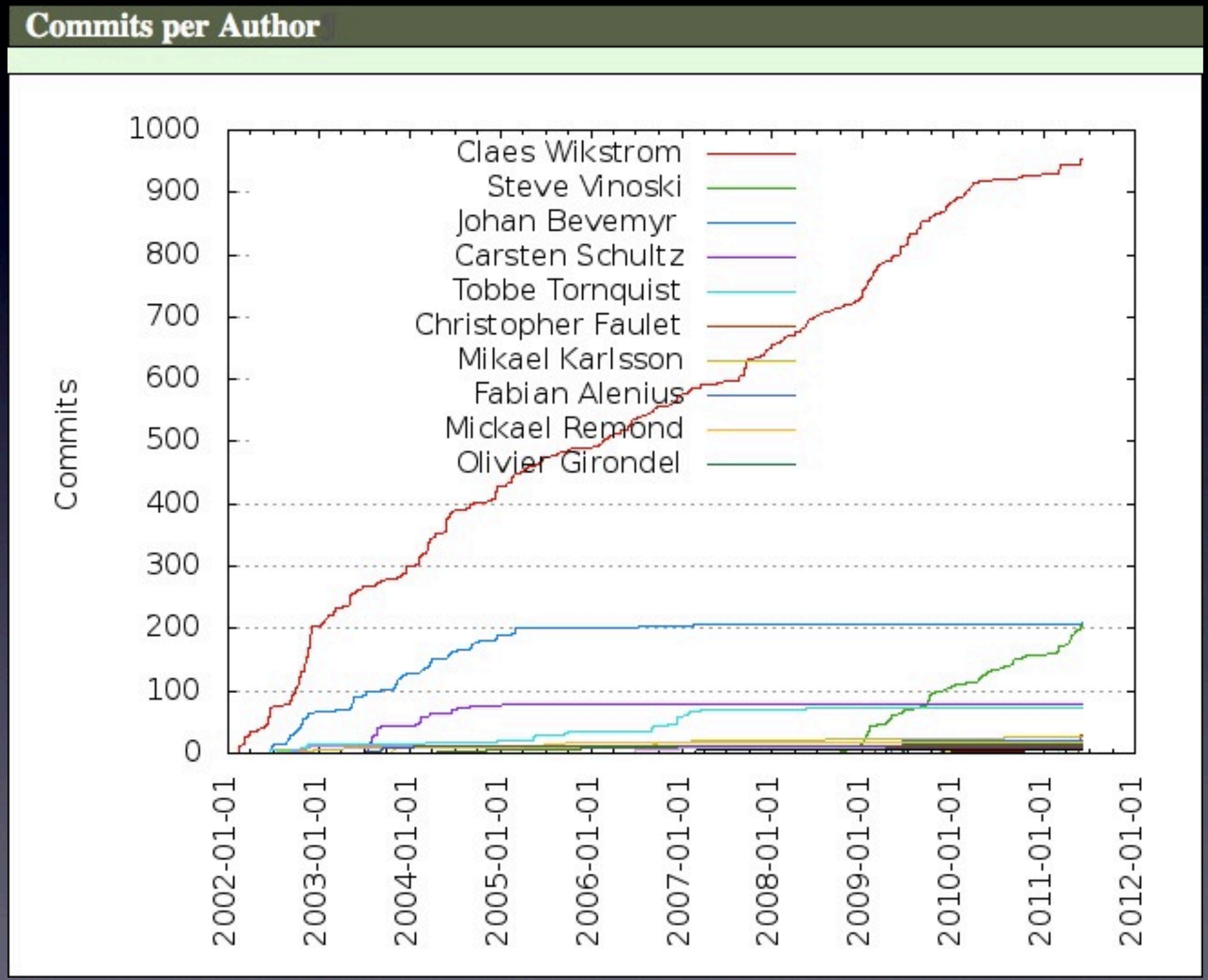
Committers



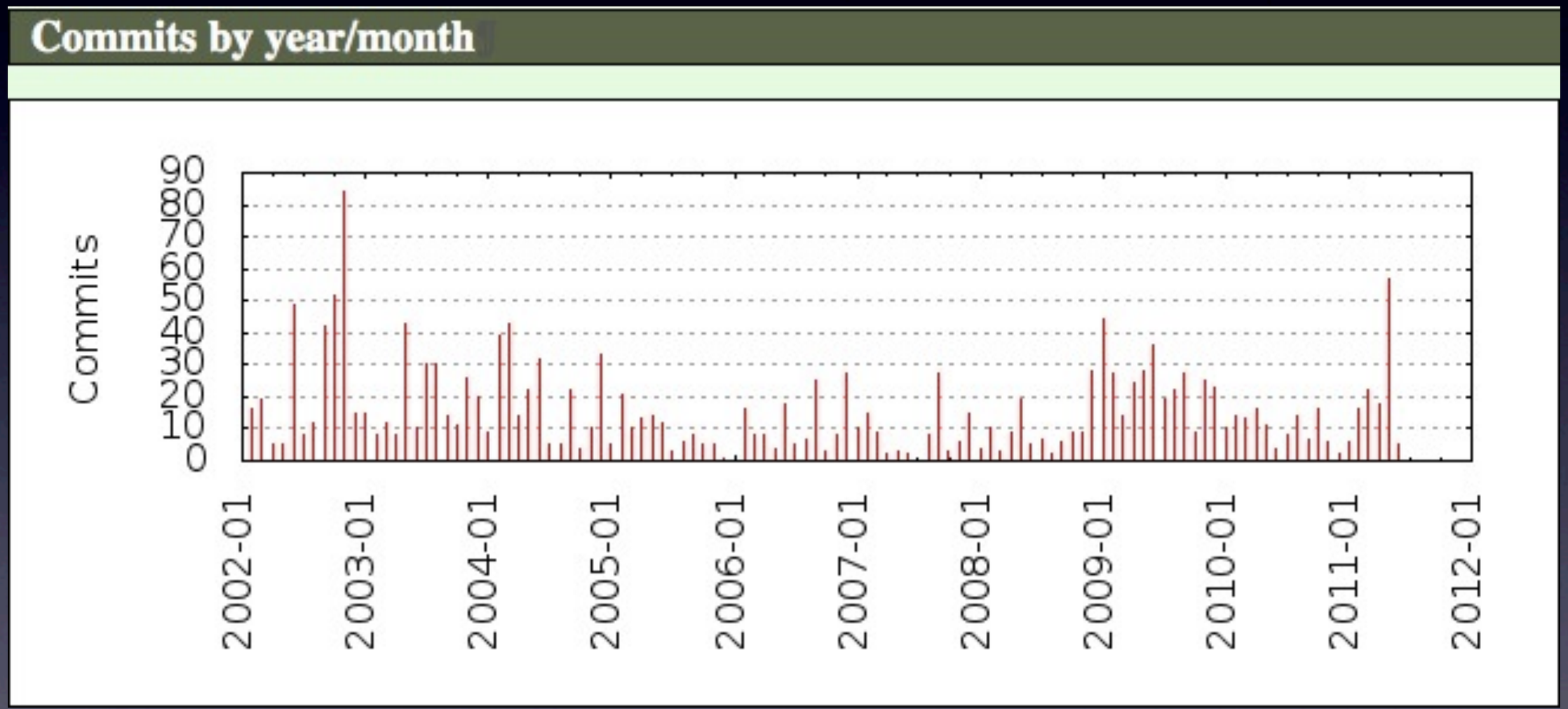
- Claes Wikstrom
- Johan Bevemyr
- Tobbe Tornquist
- Mikael Karlsson
- Mickael Remond
- Luke Gorrie
- Martin Bjorklund
- Tuncer Ayaz
- Davide Marquês
- Julian Noble
- Eric Liang
- Nicolas Thauvin
- Thomas O'Dowd
- Peter Lemenkov
- Fabian Linzberger
- Anders Nygren
- Steve Vinoski
- Carsten Schultz
- Christopher Faulet
- Fabian Alenius
- Olivier Girondel
- Leon Smith
- Yariv Sadan
- Hans Ulrich Niedermann
- Paul Hampson
- Sean Hinde
- Per Andersson
- Dominique Boucher
- Sebastian Stroll
- Jean-Sébastien Pédron
- Bruce Fitzsimmons

(two or more commits)

Commits per Committer



Commits per Month



Top Contributors

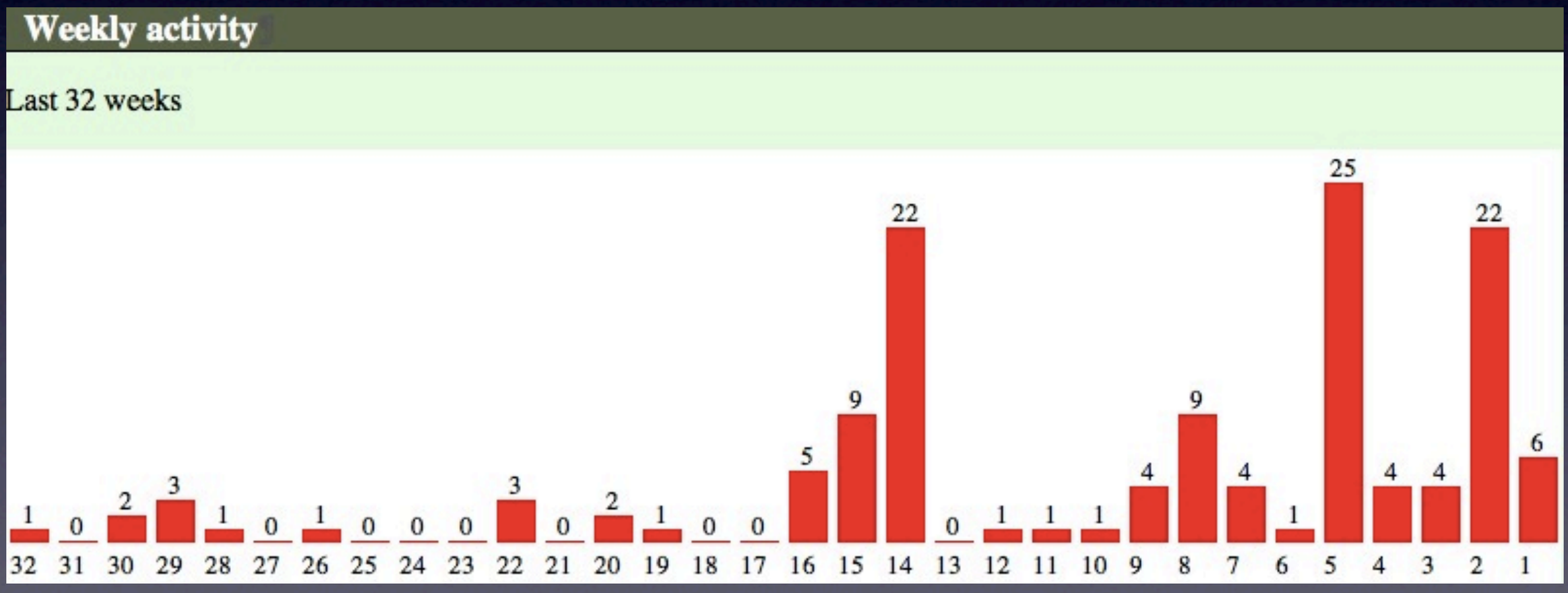
Author of Year

Year	Author	Commits (%)	Next top 5	Number of authors
2011	Steve Vinoski	52 (41.94% of 124)	Christopher Faulet, Claes Wikstrom, Tuncer Ayaz, Torbjorn Tornkvist, Nicolas Thauvin	9
2010	Steve Vinoski	50 (41.32% of 121)	Claes Wikstrom, Mikael Karlsson, Torbjorn Tornkvist, Per Andersson, Dominique Boucher	16
2009	Claes Wikstrom	141 (47.32% of 298)	Steve Vinoski, Fabian Alenius, Olivier Girondel, Hans Ulrich Niedermann, davide	12
2008	Claes Wikstrom	88 (79.28% of 111)	Steve Vinoski, Tobbe Tornquist, Mikael Karlsson	4
2007	Claes Wikstrom	80 (80.00% of 100)	Tobbe Tornquist, Julian Noble, Mikael Karlsson, Johan Bevemyr	5
2006	Claes Wikstrom	84 (65.12% of 129)	Tobbe Tornquist, Yariv Sadan, Mikael Karlsson, Johan Bevemyr, Sebastian Stroll	7
2005	Claes Wikstrom	62 (60.19% of 103)	Tobbe Tornquist, Johan Bevemyr, Mickael Remond, Martin Bjorklund, Carsten Schultz	7
2004	Claes Wikstrom	128 (53.78% of 238)	Johan Bevemyr, Carsten Schultz, Martin Bjorklund, Tobbe Tornquist, Leon Smith	8
2003	Claes Wikstrom	98 (43.17% of 227)	Johan Bevemyr, Carsten Schultz, Leon Smith, Mickael Remond, Tobbe Tornquist	8
2002	Claes Wikstrom	203 (66.12% of 307)	Johan Bevemyr, Tobbe Tornquist, Luke Gorrie, Mikael Karlsson, Seah Hinde	7

A total of 109 individual contributors

Project Activity

- Last 32 weeks



Yaws Features

Features and Concepts

A large number of features

built over

A small number of concepts

- At its core Yaws is pretty straightforward
- Event handling and dispatch in yaws_server.erl
- Yaws makes simple apps simple, and feature-rich apps possible

Configuration

- Two configuration methods:
  - config file, for stand-alone server
  - records and lists, for embedded server
- Two levels of config:
  - global, via #gconf record
  - per virtual server, via #sconf record

File Serving

- Use target URI path together with configured document filesystem root path to locate file
- Yaws uses a driver for the sendfile() system call on platforms that support it
- lower CPU, fewer system calls
- Tuncer Ayaz now pulling sendfile driver into Erlang/OTP

Dynamic Apps

- Only serving files? Use G-WAN or nginx
- Use Yaws for generating and serving dynamic content
- Take advantage of multiple apps per VM, supervisors, behaviors, massive concurrency, reliability

“.yaws” Pages

- Intermix HTML and Erlang
- Enclose Erlang code between <erl>...</erl> tags
- To serve a “.yaws” page, Yaws:
  - JIT-compiles the code (and caches it)
  - runs it (expects to find an out/1 fun)
  - replaces <erl>...</erl> with the out/1 result
- JIT page compilation was in Yaws from day one

ehtml

- HTML encoded as Erlang terms
- <tag attr1=”attr1val” attr2=”attr2val”>
  child
  </tag>

  is represented in ehtml as the Erlang term

  {tag, [{attr1, “attr1val”},{attr2,”attr2val”}],
  “child”}

#arg

- Key Yaws data structure
- Contains everything Yaws knows about a request
  - client socket, HTTP headers, request info, target URL details
- Yaws passes #arg to application code and callbacks
- See http://yaws.hyber.org/arg.yaws for details

“.yaws” Example

- <html>
  <erl>
  out(_Arg) ->
  {ehtml,
  [{title, [], “Hello World”},
  {p, [], “Hello, world!”}]}.
  </erl>
</html>

out/1

- Several Yaws features require an app-provided out/1 callback fun, for example
  - ".yaws" pages
  - appmods (application modules)
  - yapps (Yaws applications)
- The single fun argument is an #arg record

Appmods

- Appmod is a module supplying an out/1 callback fun taking an #arg record
- Appmods are configured for URL paths or sub-paths
- When a request arrives for a matching URL, Yaws dispatches the request to the appmod callback fun

Appmod Origins

- Appmods appeared around 2004
- Klacke was at Nortel, and used Yaws as an embedded server in an SSL VPN product
- Needed full control over the URL, which appmods provided
- Very versatile — probably the most used feature of Yaws

Appmod Config

- Example server appmod config (in config file):

  appmods = <cgi-bin, yaws_appmod_cgi>

- This attaches the yaws_appmod_cgi module to any cgi-bin portion of the target URL
- You can have multiple appmods per server
- Attach an appmod to “/” to control all URLs under a given server

Streaming Data

- Server might not know the response size, might not yet have all the data, might not want full response data to be all in memory
- HTTP 1.1 chunked transfer encoding helps with these cases
- Yaws also allows app processes to stream, useful for long-polling apps or streaming of non-chunked data

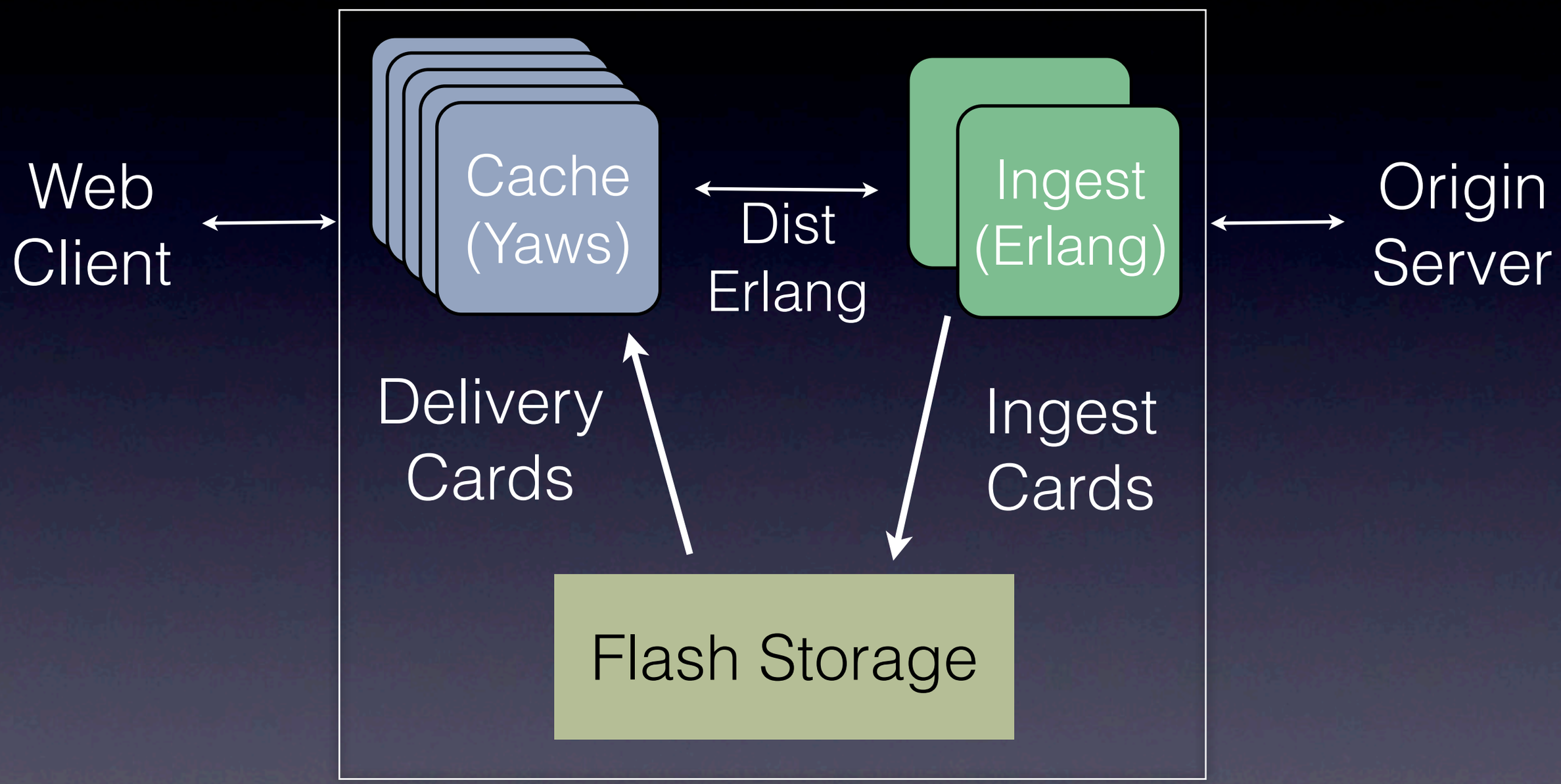
Chunked Transfer

- out/1 first returns {streamcontent, MimeType, FirstDataChunk}
- Yaws then awaits further chunks from the app
- App sends subsequent chunks via yaws_api:stream_chunk_deliver(Pid, Chunk)
- App ends transfer with yaws_api:stream_chunk_end(Pid)

Streaming Process

- out/1 can return {streamcontent_from_pid, MimeType, Pid}
- Pid identifies process streaming data back to client
- Yaws yields client socket control to Pid
- To send data, Pid calls yaws_api:stream_process_deliver(Socket, IoList)
- To finish, Pid calls yaws_api:stream_process_end(Socket, YawsPid)

Streaming Example



Verivue MDX 9000 video delivery hardware

Yaws Integration

- Linked-in driver for open, close, and sockopts
- Used with gen_tcp via {fd, Fd} option
- Tracked by controlling process
- Yaws integration via fdsrv module and streaming API
  - fdsrv enabled integration with offload listen socket
- ***Zero changes*** to Yaws needed in order to use it in this application

Embedding

- Some think Yaws is only a stand-alone server
- For a stand-alone server, Yaws is the top-level or “controlling” app
- But Yaws can also be embedded within or beside other applications
  - start under your own supervisor
  - or start as an embedded app

Embedding Under Your Supervisor

- yaws_api:embedded_start_conf(Docroot, ServerConfList, GlobalConfList, ServerId)
- Returns full global and server configurations along with child specs for Yaws processes
- Start the children under your supervisor
- Then call yaws_api:setconf with the full global and server configurations

Other Yaws Features

- HTTP 1.1
- URL/#arg rewriting
- Yaws applications (yapps)
- SSL support
- cookie/session support
- munin stats
- CGI and FCGI
- reverse proxy
- file upload
- WebDAV
- small file caching
- SOAP support
- haXe support
- JSON and JSON-RPC 2.0
- websockets
- GET/POST chunked transfer
- multipart/mime support
- file descriptor server (fdsrv)
- server-side includes
- heart integration
- both make and rebar builds
- man pages
- LaTex/PDF documentation

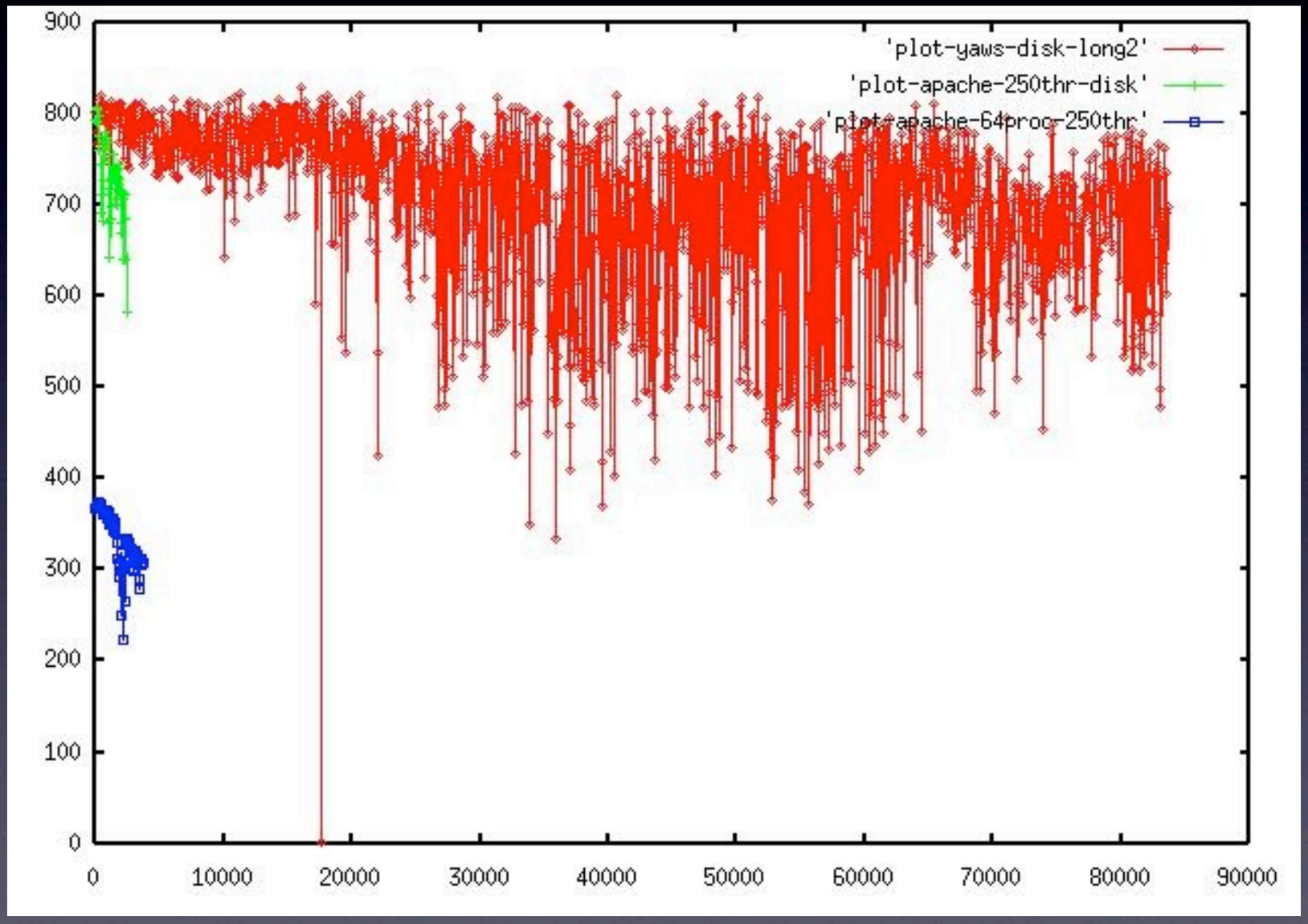
Recent Work

- Major contributions from Christopher Faulet for version 1.90
- Examples: authorization improvements, ACLs, config for Expires header, more flexibility for #arg rewrite modules, better shutdown control, traffic shaping, extended PHP handler
- Nearly 30 changes/features in all

Performance

Apache vs. Yaws

http://www.sics.se/~joe/apachevsyaws.html



But Ironically...

```
$ curl -I http://www.sics.se/~joe/apachevsyaws.html

HTTP/1.1 200 OK
Date: Fri, 03 Jun 2011 02:06:59 GMT
Server: Apache/2.2.6 (Unix)
Accept-Ranges: bytes
Content-Length: 4286
Connection: close
Content-Type: text/html
```

The Performance Presumption

- Why do people benchmark requests per second? Because it's easy.
- Many seem to presume/conclude the ***fastest*** server is the ***best*** server
- Benchmarks often completely miss actual application concerns
- A 2003 article: http://steve.vinoski.net/pdf/IEEE-The_Performance_Presumption.pdf

Benchmarking Advice

- Great advice from Mark Nottingham's blog: http://www.mnot.net/blog/2011/05/18/http_benchmark_rules
- More great advice from Jesper Louis Andersen's blog: http://jlouisramblings.blogspot.com/2011/06/web-server-benchmarking-rant.html

mnot's Advice

- Consistency: use same OS, hardware, network, set of running apps every time, and avoid virtual machines
- Keep test clients off the server host
- Understand limiting factors such as network bandwidth
- Tune the OS
- Avoid short duration "hello world" tests
- Follow the link for more advice

JLouis's Advice

- Pay attention to handling overload
  - "*...for most servers, the speed is so good it doesn't matter. Stability and the overload situation is more important to optimize for.*"
- Pay attention to latency and outliers (stability)
- Beware the average (same advice from mnot as well)

My Advice

- Don’t believe benchmarks posted by server developers — *do your own benchmarking*
- Beware of servers that don’t fully support HTTP (achieving “speed” by leaving out critical support)
- In addition to other problems, req/sec measurements on “hello world” tests ignore differences in HTTP header sizes, resulting in apples-oranges comparisons

Bottlenecks

- Event loops, socket handling, data handling don’t vary much across well-written Erlang web servers
- Erlang web server performance is significantly impacted by the VM
  - TCP driver
  - HTTP packet decoding

The Lisp Curse

- "*Lisp is so powerful that problems which are technical issues in other programming languages are social issues in Lisp.*"

  http://www.winestockwebdesign.com/Essays/Lisp_Curse.html

- "*Exercise for the reader: Imagine that a strong rivalry develops between Haskell and Common Lisp. What happens next?*

  Answer: The Lisp Curse kicks in. Every second or third serious Lisp hacker will roll his own implementation of lazy evaluation, functional purity, arrows, pattern matching, type inferencing, and the rest. Most of these projects will be lone-wolf operations."

The Erlang Curse?

- Erlang makes writing network apps and servers so easy, pretty much anyone can do it
- It's too easy to
  - just claim that existing systems like Yaws are too complicated, without really trying them
  - and roll your own "lightweight" server instead
- Do we really need more Erlang web servers?
- And what about community fragmentation?

Advancing the State of the Art

- You want to make Erlang web servers significantly faster? Write a HTTP TCP driver we can all use

- Or focus on programming models, like webmachine, nitrogen, EWGI have done

- Or if you seek server modularity, talk to us. Klacke and I gladly welcome questions, observations, suggestions, and contributions

Yaws Future

Is Yaws “Cool”?

- Yaws isn’t cool if
  - you judge coolness by how new something is
  - you judge coolness by the project’s website :-)
- Yaws ***is*** cool if
  - years of reliability and stability are cool
  - a decade-old, still very active and widely deployed open source project with useful features and a strong community of users and contributors and is cool

Going Forward

- Continue accepting community contributions
- Keep an eye on HTTP changes (like PATCH)
- Keep an eye on developing standards like websockets
- Reverse proxy work, possible changes to dispatching for more flexibility/modularity
- More and better tests
- Version 2.0?
- Above all, continue offering great stability, reliability, performance and scalability

Thanks